

---

**VueCentric**

---

# **Technical Specification**

**Version 1.0**

---

# Table of Contents

Table of Contents .....	2
Introduction .....	5
Architecture .....	5
Visual Interface Manager .....	8
Command Line Parameters .....	8
autologin .....	8
blank .....	9
caption .....	9
debug .....	9
host=<hostname>:<port> .....	9
icon=<filename> .....	9
image=<filename> .....	9
noccow .....	9
nocompose .....	10
nodesign .....	10
nojoin .....	10
notimeout .....	10
nouupdate .....	10
port=<port number> .....	10
server=<hostname>:<port> .....	10
showflags .....	11
template=<template> .....	11
updateall .....	11
verbose .....	11
VIM Automation Object .....	11
Properties .....	12
BringToFront .....	12
Popup .....	12
Component Support Services .....	14
Server Automation Object .....	14
Properties .....	14
Session Automation Object .....	14
Properties .....	15
CallRPCAbort .....	15
CallRPCAsync .....	16
CallRPCBool .....	16
CallRPCDate .....	17
CallRPCInt .....	17
CallRPCList .....	18
CallRPCStr .....	18
CallRPCText .....	19
CCOWJoin .....	19
CCOWLeave .....	19

Connect.....	20
ContextChangeBegin .....	20
ContextChangeEnd.....	20
Disconnect.....	20
FindObjectByCLSID.....	21
FindObjectByIID.....	21
FindObjectByProgID .....	22
FindServiceByCLSID .....	22
FindServiceByProgID .....	22
EventFireLocal.....	23
EventFireRemote.....	23
EventSubscribe.....	23
EventUnsubscribe.....	24
RegisterObject.....	24
ReplaceParams .....	24
UnregisterObject .....	25
ICSS_SessionEvents.....	26
EventCallback .....	26
RPCCallback.....	26
RPCCallbackError.....	26
ICSS_Context.....	27
Properties.....	27
CommitContext.....	27
GetContext .....	27
Init .....	28
Reset.....	28
SetContext.....	28
User Context Object .....	28
Properties.....	29
HasKey .....	29
HasKeys .....	30
ESigValidate.....	30
Patient Context Object.....	31
Properties.....	31
Encounter Context Object .....	32
Properties.....	33
Prepare.....	33
Site Context Object.....	34
Properties.....	34
Context Change Events .....	34
Pending.....	35
Committed.....	35
Canceled.....	35
Global Object Registry .....	35
Template Registry.....	37
TObjectContainer.....	38

TPanelEx .....	40
TScrollBarEx .....	40
TLabelEx .....	41
TToolBarEx .....	42
TPageControlEx .....	42
TTabSheetEx .....	43
TSplitterPaneEx .....	43
TPaneEx .....	44
TTreeViewEx .....	45
TTreePaneEx .....	46
Object Repository .....	48
Glossary .....	49
MIDL Specification .....	50
Visual Interface Manager .....	50
Component Support Services .....	51

## Introduction

VueCentric is a multi-tiered, component-based clinical desktop application that supports a wide range of clinical functions using standardized, plug-in objects. With the appropriate server-side VistA/RPMS components, the fully implemented version has objects that support patient lookup, clinical encounter documentation, on-line ordering, results retrieval, decision support, problem list management, consult tracking and adverse reaction tracking, to list a few. Using a Visual Interface Manager (VIM), power users may select from a palette of objects and construct a fully functional application from discrete components. An application can be designed to meet the needs of an individual, user class, site, or a specialized requirement. Once assembled, a configuration has the appearance of a cohesive whole, belying its component-based origins. Each component communicates with a middle tier Component Support Services (CSS) that coordinates the activities of the objects so that the result is a seamless application. The CSS provides event and context management and remote data access services to components within its application space. The CSS also communicates with any CCOW-compliant context manager to allow the application and its components to share context state with other CCOW-aware applications running on the same desktop.

## Architecture

Before one can support the interoperability of plug-in components, one must explicitly define the rules for such interoperability. VueCentric defines a multi-tiered architecture that insures the interoperability of components developed in accordance with the specification. From top to bottom, the tiers are the Visual Interface Manager (which acts as a container and facilitator for the individual components), the Component Support Services (which provide context and event management and access to shared resources), and the underlying host system (currently DHCP and RPMS, but other host systems are possible as well).

The *Visual Interface Manager* (VIM) acts as the glue that holds the individual components together. It empowers the user to define the visual relationships among discrete components, provides the ability to compose complex interfaces from individual visual elements, supports the streaming of compositional entities to and from a central store, controls user-level access to components, and can interrogate components for the resources they require and automatically connect them to those resources.

The *Component Support Services* (CSS) provide shared resources that all components may access and coordinate the activities of components. The CSS supports the concept of plug-in services that extend the functionality of the middle tier in a fully extensible manner. Available services include context objects that reflect the current state of the application, such as the currently selected patient, the user who is logged in, or the clinical encounter that is being referenced. Other plug-in services include unified signature, remote data views and clinical reminders. The CSS also provides support for performing remote procedure calls to allow objects to interact with the host system. The CSS is also a manager and producer of events that

can notify components who choose to subscribe that, for example, the patient selection has changed. Finally, the CSS can also participate in context changes that originate outside the application. Because the CSS automatically detects the presence of a CCOW-compliant context manager and registers as a participant, the VueCentric application may synchronize its context with other CCOW-compliant applications residing on the same desktop.

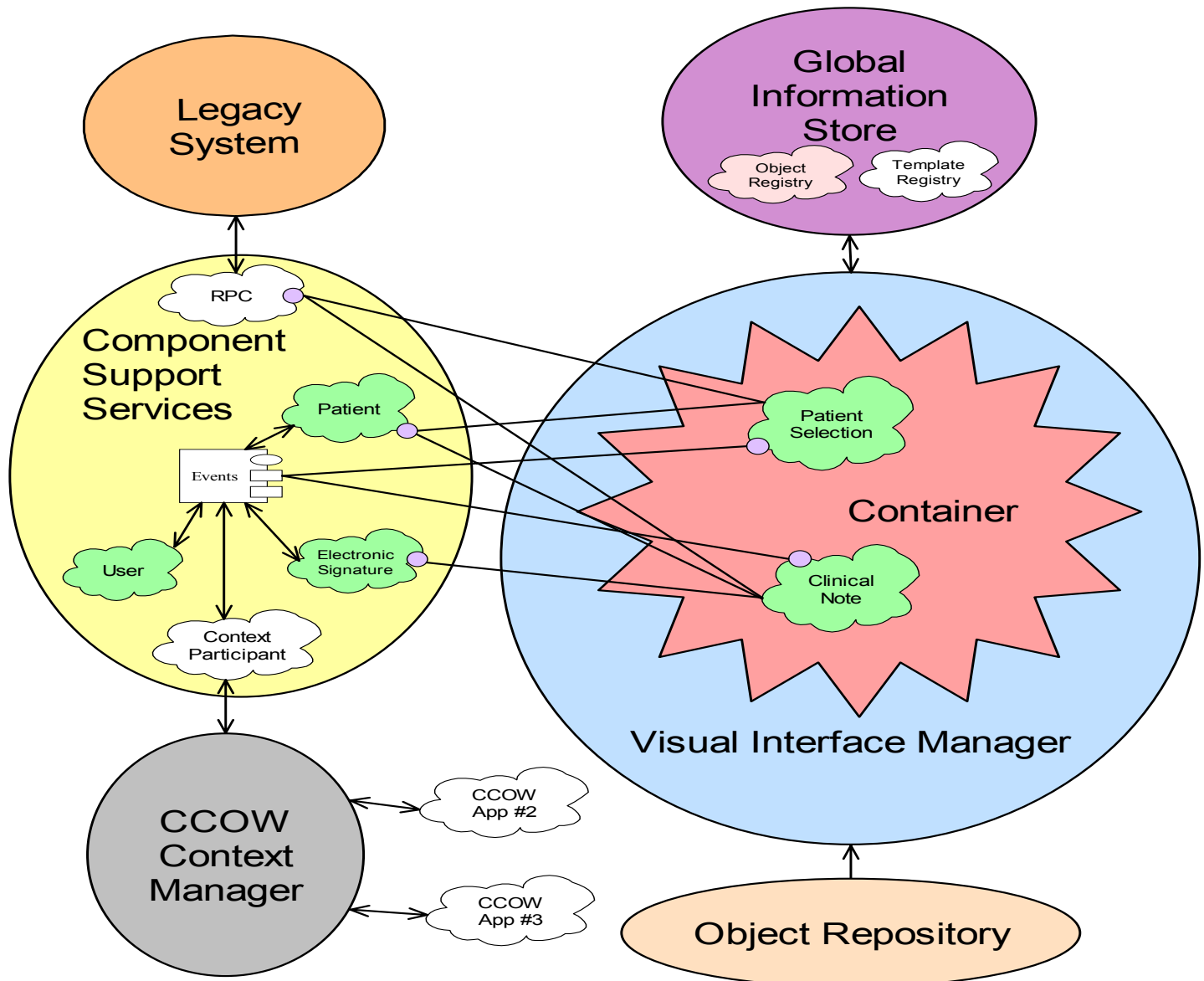
While the target *host systems* for VueCentric will initially be DHCP and RPMS, the architecture attempts to hide the origins of the host data from the consumers of these data (i.e., the components) by encapsulating the access methods within the CSS. The degree to which this is successful will largely be determined by our ability to devise common information models for these data that can be mapped to different host representations. Much of the effort that remains is the definition of a common information model for entities such as patient, user, and encounter. There is much activity in this area within VHA (G-CPR) and without (CORBAMed, MSHUG, HL7 RIM). It is our aim to leverage these efforts to evolve this specification.

In addition to the three tiers described above, the VueCentric architecture also includes external data stores in the form of an *Object Registry*, a *Template Registry*, and an *Object Repository*.

The *Object Registry* provides information about available objects and their default characteristics.

The *Template Registry* provides a globally accessible location for the storage of state-information in a context-free format. It is used to store user interface configuration information.

The *Object Repository* is a store of components that are accessible to the VueCentric application. The component repository allows an application to automatically update locally installed components from a reliable source. The component repository may be implemented by a web server, an ftp server, or any globally accessible directory, or any combination of all three. The VueCentric architecture models the updating of components after that employed by web browsers. Under that paradigm, an HTML document requests a component by a unique identifier and version. If the requested component is already available locally, that version is used. If it is not, the HTML document includes a URL reference that defines a source from which the component may be downloaded and installed. The VueCentric application uses a very similar approach that permits it to automatically propagate updates to existing components as well as new components to individual workstations as they are needed.



## The *VueCentric* Architecture

# Visual Interface Manager

The Visual Interface Manager (VIM) provides a number of services:

- Acts as an intelligent container for components
- Provides access to persistent state information
- Defines the visual relationship of components to one another
- Possesses a design feature that allows the tailoring of the environment under user control
- Initializes and prepares the Component Support Services for use by components
- Provides menu management

When the user logs in, the VIM accesses the Object and Template Registries residing on the host system to retrieve information about the requested configuration. Using this information, the VIM reconstructs the visual interface. The requested configuration may be the user's private configuration or one that is defined for a specific user role or function. Specific templates may be requested using the appropriate command line parameter when invoking the VIM. In the absence of such, the VIM retrieves the user's customized template if one exists or a default template that is determined by the host configuration and can be specific to the user, user class, department or institution.

Before each component is loaded into the visual interface, the VIM references the Object Registry to determine if the object is available and if the user is permitted to use it. If the latest version of the object is currently installed, the VIM retrieves it from the Object Repository. If all of these conditions are met, the VIM instantiates a copy of the component, performs any initializations specified by the Object Registry, and restores any saved state information (e.g., the color of the component as set by the user). The VIM then registers the component with the Component Support Services (CSS). The purpose of the registration process is to allow the CSS to determine if the component implements any of the event interfaces it or its plug-in services publishes. For each implemented interface, the CSS connects the interface to the corresponding event producer. Thus, all a component must do to subscribe to an event is to simply implement the corresponding interface. The registration process takes care of the rest.

## **Command Line Parameters**

The VIM executable recognizes a variety of command line parameters. Some of these are provided to facilitate object development and testing. Command line parameters are case-insensitive may be preceded by a hyphen or forward slash, but neither is required. Some parameters accept a value, which should be separated from the parameter by an equal sign (without leading or trailing spaces). Parameter values with imbedded spaces must be surrounded by quotation marks. The VIM recognizes the following parameters

*autologin*

Automatically prompts for user login upon startup. This differs from the default behavior where the VIM initially presents a blank desktop and the user clicks the desktop to initiate login. In the default mode, the user is returned to the blank desktop upon logging out. With *autologin* enabled, the user is immediately presented with an authentication request upon starting the VIM and the VIM terminates upon logging out.

***blank***

Suppresses the loading of a template upon login. Instead, the VIM presents the user with a blank desktop. This is useful for testing objects and for designing templates where an initial configuration is not desired.

.

***caption***

Sets the caption for the application's main form. The caption may also be set in design mode as a desktop property and saved as a template.

***debug***

Sets the CSS to debug mode. This activates debug mode for the RPC broker and enables a special trace mode that causes the CSS to output remote procedure call data via the TRACE event. Any object subscribing to the TRACE event may view this data.

***host=<hostname>:<port>***

Sets the name of the target host system. This is identical to the *server* command line parameter. See the description of that parameter for more details.

***icon=<filename>***

Allows the specification of an icon file. When specified, the contained icon will be used in place of the application's default icon. This can also be set as a desktop property when in design mode and saved as part of a configuration.

***image=<filename>***

Allows the specification of an image file. When specified, the contained image (several image formats are supported) is displayed in place of the blank desktop when the VIM is in the logged out state. This can also be set as a desktop property when in design mode and saved as part of a configuration.

***noccow***

Disables CCOW interaction. When this parameter is specified, the VIM does not instruct the CSS to attempt to contact a CCOW context manager even if one is present. This is useful for

debugging and when a second instance of the VIM is desired that does not share context with other applications.

***nocompose***

Disables compose mode even if user possesses the necessary security key. This is provided primarily for debugging purposes.

***nodesign***

Disables design mode even if user possesses the necessary security key. This is provided primarily for debugging purposes.

***nojoin***

Disables automatic joining of CCOW context. By default, the VIM instructs the CSS to connect to a CCOW context manager (if one is detected and the *noccow* command line parameter is not specified) and join the common context. If *nojoin* is specified, the CSS still connects to the context manager, but does not join the context. Unlike the *noccow* parameter, the user still retains the option of manually joining the common context by right-clicking the CCOW icon in the lower right corner of the application window and selecting the appropriate popup menu choice.

***notimeout***

Disables auto timeout. By default, the VIM logs out after a site-specified period of keyboard and mouse inactivity. This option causes the VIM to ignore the timeout.

***noupdate***

Disables the automatic updating of objects. When this parameter is specified, the VIM does not retrieve objects from the Object Repository. This parameter is provided primarily for debugging purposes. It may also have application in environments that use system administrator tools to push software updates to workstations.

***port=<port number>***

Sets the port number for the host connection. Note that the port number may also be specified as part of the *server* command line parameter.

***server=<hostname>:<port>***

Specifies information about the target host. If this parameter (or the functionally equivalent *host* parameter) is not specified, the CSS will either select the default host or present the user with a choice of hosts. Which action is taken depends upon how the RPC broker has been configured on the workstation. If the port number is omitted, the default port for the host will be used.

***showflags***

Shows active command line flags in the status panel. This is useful for debugging purposes to have a visual indicator of which command line flags are in effect.

***template=<template>***

Loads the named template upon login. By default, the VIM loads the user's personal configuration template upon login or, in the absence of one, the user's default template as determined by site configuration. This parameter overrides this default behavior.

***updateall***

Overrides the default behavior of updating objects only if they are not present or newer versions are available in the Object Repository by forcing updates to occur every time an object is requested. This is provided primarily for debugging purposes.

***verbose***

Displays additional status information for debugging.

**VIM Automation Object**

Components typically have minimal interaction with the VIM. Rather, they react passively to control by the container. The component's principal interaction is with the CSS or another component. The VIM does, however, register an automation interface with the CSS that is accessible by components. This interface is defined as follows:

Programmatic Id: CIA\_VIM.VIM  
Class GUID: A45DCEDF-22F6-4F2B-BA12-DF5D5765ED68  
Default Interface: IVIM

***Properties***

Property	Datatype	Access	Description
Caption	WideString	RW	The caption of the application's main form.
Font	IFontDisp	RW	The default font for the application. Changes to this property are automatically propagated to all objects that also publish a font property.
Icon	WideString	RW	The name of the file that contains an icon to be displayed in place of the application's default icon. If null, the application's default icon is used.
Image	WideString	RW	The name of the file that contains an image to be displayed while the application is in a logged out state. If null, no image is displayed.

Interface Methods:

***BringToFront***

Parameter	Datatype	Description
ObjRef	IUnknown	IUnknown interface reference of the object to be brought to the foreground.
<return value>	WordBool	Returns true if the referenced object was found.

This function may be invoked to insure that the referenced object is visible within the visual interface. If other windows obscure the object, it is brought to the top of the Z-order. If the object is located on a tab or pane that is not currently active, that tab or pane will be automatically activated.

***Popup***

Parameter	Datatype	Description
ObjRef	IUnknown	IUnknown interface reference of the object to be displayed modally.
Title	WideString	The caption to be displayed for the popup dialog.
<return value>	WordBool	Returns true if the referenced object was found.

This function causes the referenced object to appear in a modal window. The VIM accomplishes this by temporarily moving the object from its parent within the interface to a modal window. When the modal window is closed, the object is returned to its original position.



## Component Support Services

The Component Support Services (CSS) provide a suite of services that allow objects to access context information (current user, current patient, current encounter, etc.), host-based data (through RPC calls) and receive event notifications (change in selected patient, for example). Unlike the VIM, which is a standalone executable, the CSS is implemented as an in-process COM server that executes in the background and is shared by all objects within an application instance. Unlike previous versions that defined a single automation server with over a dozen interfaces, the CSS implements two automation servers and defines four interfaces. Many of the interfaces previously declared by the CSS that provided access to context information are implemented as separate automation objects that are registered with the CSS at runtime.

The two automation servers defined by the CSS are the Server (CIA\_CSS.CSS\_Server) and the Session (CIA\_CSS.CSS\_Session) objects. The sole purpose of the Server object is to instantiate and maintain a shared instance of the Session object. A component cannot directly create an instance of the Session object. Rather, it must first create an instance of the Server object and request a reference to the Session object. Having obtained such a reference, the component has no further need of the Server object and may release it.

### **Server Automation Object**

The server automation object has the following definition:

Programmatic Id:     CIA\_CSS.CSS\_Server  
Class GUID:         8C061A95-8FCE-41A7-A806-66B02E5CE6EF  
Default Interface:   ICSS\_Server

#### ***Properties***

Property	Datatype	Access	Description
Session	ICSS_Session	R	Reference to the session automation object.

Components desiring access to middle tier services should do so by first creating an instance of the server automation object and then obtaining a reference to the session automation object by reading the value of the Session property. Once this is done, the reference to the server object can be released.

### **Session Automation Object**

The session automation object provides access to middle tier services. While the session object has a programmatic identifier and class GUID, it cannot be instantiated directly, but must be requested from the server automation object. The session automation object has the following definition:

Programmatic Id: CIA\_CSS.CSS\_Session  
 Class GUID: 8C061A95-8FCE-41A7-A806-66B02E5CE6EF  
 Default Interface: ICSS\_Session  
 Event Interface: ICSS\_SessionEvents

### ***Properties***

Property	Datatype	Access	Description
CCOWState	Enum	R	The current CCOW state. Possible values are: ccowBroken = Not participating ccowChanging = Context change in progress ccowJoined = Participating ccowNone = No context manager ccowDisabled = Disabled by host
DebugMode	WordBool	RW	Indicates whether or not the CSS is in debug mode.
DomainName	WideString	R	The domain name of the currently connected host. If there is no active connection, returns null.
HostAddress	WideString	R	The IP address of the host system that is currently connected. If no connection is active, returns null.
HostDateTime	TDateTime	R	The current date and time as returned by the host system. If there is no active connection, returns a NULL_DATE value.
HostName	WideString	R	The name of the host system that is currently connected. If no connection is active, returns null.
HostPort	Integer	R	The port number of the active connection. If no connection is active, returns 0.
Param(Name)	OleVariant	RW	Allows an object to register an arbitrary named parameter and value that can be accessed by other objects and by the <i>ReplaceParams</i> method. The Name parameter may consist only of alphanumeric characters and the underscore.

### ***CallRPCAbort***

Parameter	Datatype	Description
Handle	Integer	Handle of the asynchronous call to abort.

This procedure causes the asynchronously executing remote procedure identified by *Handle* to be aborted.

***CallRPCAsync***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
Callback	ICSS_SessionEvents	Callback interface to be invoked on completion of the remote procedure.
PlainText	WordBool	If true, data returned to the callback interface is in plain text format. Otherwise, format is in comma text format.
<return value>	Integer	Handle that uniquely identifies this asynchronous call.

This function invokes the remote procedure named in *RPCName* in asynchronous mode, passing it the parameters listed in *Parameters* (see description of *CallRPCList* for details on formatting of parameters). A negative return value indicates that the remote procedure failed. Otherwise, the return value is a unique handle that identifies the call. Upon completion of the remote procedure, the callback interface identified by the *Callback* parameter is invoked. See a description of the *ICSS\_SessionEvents* interface for details.

***CallRPCBool***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	WordBool	Output of remote procedure call as a Boolean value.

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters* (see description of *CallRPCList* for details on formatting of parameters). The return value is a Boolean result.

***CallRPCDate***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	TDateTime	Output of remote procedure call as a TDateTime datatype.

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters* (see description of *CallRPCList* for details on formatting of parameters). The return value is a TDateTime datatype.

***CallRPCInt***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	Integer	Output of remote procedure call as a 32-bit integer value.

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters* (see description of *CallRPCList* for details on formatting of parameters). The return value is a 32-bit integer value.

***CallRPCList***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	WideString	Output of remote procedure call in CommaText format.(see discussion).

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters*. The return value is a string in CommaText format. This format can be converted to a TStringList descendant by setting it into the CommaText property.

The *Parameters* argument may be any OleVariant datatype, including a variant array. If it is scalar (non-array) datatype, it is passed as a single argument to the remote procedure call. If it is an array, each element of the array is passed as an argument. If an array element is itself an array, it is passed as a list argument to the remote procedure call.

```

var
  lstXYZ: TStringList;
begin
  lstXYZ:=TStringList.Create;
  lstXYZ.CommaText:=Shell.CallRPCList('FETCH',[Name,SSN]);
  ...
end;
```

***CallRPCStr***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	WideString	Output of remote procedure call as a string.

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters* (see description of *CallRPCList* for details on formatting of parameters). The return value is a string.

***CallRPCText***

Parameter	Datatype	Description
RPCName	WideString	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter.
Parameters	OleVariant	Parameters to be passed to the remote procedure.
<return value>	WideString	Output of remote procedure call in plain text format.(see discussion).

This function invokes the remote procedure named in *RPCName*, passing it the parameters listed in *Parameters*. The return value is a string in plain text format. This format can be converted to a TStrings descendant by setting it into the Text property.

The *Parameters* argument may be any OleVariant datatype, including a variant array. If it is scalar (non-array) datatype, it is passed as a single argument to the remote procedure call. If it is an array, each element of the array is passed as an argument. If an array element is itself an array, it is passed as a list argument to the remote procedure call.

***CCOWJoin***

Parameter	Datatype	Description
<return value>	WordBool	True if the CSS was successful in joining the common context.

This function instructs the CSS to contact a CCOW-compliant context manager and register itself as a context participant. If no context manager is present, CCOW support has been disabled, or the attempt to join the common context failed, the function returns false.

This function is reserved for use by the VIM.

***CCOWLeave***

This parameterless procedure causes the CSS to suspend its participation in the CCOW context. If no context manager is present, or the CSS is not an active participant, no action is taken.

This function is reserved for use by the VIM.

***Connect***

Parameter	Datatype	Description
Server	WideString	Name of the remote system to be connected. The format is: <username>:<password>@<hostname>:<port> Any portion may be omitted. If the hostname is omitted, either the default host is selected or a list of available hosts is presented, depending on the workstation configuration. If the port is omitted, the host's default RPC port is used. If username and password are omitted, the host requests authentication.
<return value>	Boolean	True if the connection request was successful.

This function connects to the remote server named in *Server*. The return value indicates the success of the request. Note that currently the CSS implements a single, shared instance of the RPC broker. This means that only a single host connection may be active at a given time. This restriction may be relaxed in future versions to allow concurrent connections to multiple hosts and, possibly, multithreaded connections to the same host.

***ContextChangeBegin***

Context objects use this parameterless procedure to initiate a context change sequence. Consecutive calls to this procedure are nested so that the context change sequence does not actually begin until an equal number of *ContextChangeEnd* procedure calls have been invoked.

***ContextChangeEnd***

This parameterless procedure decrements the context change reference count and invokes a context change sequence when the reference count reaches zero.

***Disconnect***

Parameter	Datatype	Description
Survey	WordBool	If true, all internal context participants for all contexts are surveyed before disconnecting. If any participant declines, the disconnect is aborted.
<return value>	WordBool	True if the disconnect was successful. This would only be false if the Survey parameter was true and a context participant declined.

This procedure terminates the connection to the remote server. If no connection is active, the call has no effect.

### ***FindObjectByCLSID***

Parameter	Datatype	Description
CLSID	GUID	The globally unique identifier of the object class to be located.
Last	IUnknown	Interface reference of the previously located interface.
<return value>	IUnknown	Returns a reference to the object implementing the class identified by <i>CLSID</i> or nil if none is found.

This function searches the list of registered objects to find one that implements the class identified by *CLSID*. If the *Last* parameter is not nil, the search begins following that object's entry in the list. In this manner, one can iterate through multiple object instances of the same class.

### ***FindObjectByIID***

Parameter	Datatype	Description
IID	GUID	The globally unique identifier of the interface to be located.
Last	IUnknown	Interface reference of the previously located interface.
<return value>	IUnknown	Returns a reference to the object implementing the interface identified by <i>IID</i> or nil if none is found.

This function searches the list of registered objects to find one that implements the interface identified by *IID*. If the *Last* parameter is not nil, the search begins following that object's entry in the list. In this manner, one can iterate through all objects implementing a particular interface.

***FindObjectByProgID***

Parameter	Datatype	Description
ProgID	WideString	The programmatic identifier of the object to be located.
Last	IUnknown	Interface reference of the previously located interface.
<return value>	IUnknown	Returns a reference to the object identified by <i>ProgID</i> or nil if none is found.

This function searches the list of registered objects to find one that possesses the programmatic identifier specified by *ProgID*. If the *Last* parameter is not nil, the search begins following that object's entry in the list. In this manner, one can iterate multiple object instances of the same class.

***FindServiceByCLSID***

Parameter	Datatype	Description
CLSID	GUID	The globally unique identifier of the service's class to be located.
<return value>	IUnknown	Returns a reference to the service implementing the class identified by <i>CLSID</i> or nil if none is found.

Request a reference to the service identified by CLSID. If the service is not already running, the CSS starts the service. If the service is not located, a nil reference is returned. Otherwise, the return value is a reference to the service's default interface.

***FindServiceByProgID***

Parameter	Datatype	Description
ProgID	WideString	The programmatic identifier of the service to be located.
<return value>	Iunknown	Returns a reference to the service identified by <i>ProgID</i> or nil if the service could not be located.

Request a reference to the service identified by ProgID. If the service is not already running, the CSS starts the service. If the service is not located, a nil reference is returned. Otherwise, the return value is a reference to the service's default interface.

***EventFireLocal***

Parameter	Datatype	Description
EventType	WideString	The name of the event type to fire.
EventStub	WideString	Additional information specific to the event type.

Broadcasts an event of type *EventType* to all subscribers within the application's process space. The effect is identical to an event generated by the host system in that callbacks are made to subscribers via the *ICSS\_SessionEvents* interface. Unlike events generated by the host system, events generated by this call are limited to subscribers within the application's process space.

***EventFireRemote***

Parameter	Datatype	Description
EventType	WideString	The name of the event type to fire.
EventStub	WideString	Additional information specific to the event type.
Recipients	WideString	Optional recipient list. If no recipients are specified, the event is broadcast to all active users on the same host.

Broadcasts an event of type *EventType* to all subscribers connected to the same host. If recipients are specified, distribution is limited to those recipients only. Unlike local events, events generated by this call are sent directly to the host system, which then redirects them to the appropriate recipients. Once an event reaches the recipient, it is further redirected to subscribers to that event within the recipient's application process space through a mechanism identical to local events (see the *ICSS\_SessionEvents* interface).

***EventSubscribe***

Parameter	Datatype	Description
EventType	WideString	The name of the event for which a subscription is desired.
Callback	ICSS_SessionEvents	A reference to the interface that will be called when an event of the specified type is received.

This method enters a subscription for the named *EventType*. The caller must specify a reference to a callback interface that will be invoked when an event of the specified type is triggered. See a description of the *ICSS\_SessionEvents* interface for more information.

***EventUnsubscribe***

Parameter	Datatype	Description
EventType	WideString	The name of the event for which a subscription is to be revoked.
Callback	ICSS_SessionEvents	A reference to the interface that was specified in the original EventSubscribe call.

This method revokes a subscription for the named *EventType*. The caller must specify a reference to the same callback interface that was specified in the original *EventSubscribe* call. Note that subscriptions are automatically revoked when an object is unregistered.

***RegisterObject***

Parameter	Datatype	Description
ObjRef	IUnknown	IUnknown interface reference of the object to be registered.

The VIM uses this procedure call to register an object with the CSS. The CSS uses the IUnknown interface reference to query the object for the interfaces it supports. When the CSS generates an event for an interface supported by the object, it performs a callback on that interface to communicate the event to the object. In this manner, objects may subscribe to an event by simply implementing the corresponding interface. The *RegisterObject* procedure takes care of connecting the object's event interface (i.e., event sink) to the event source.

***ReplaceParams***

Parameter	Datatype	Description
Source	WideString	The value to be parsed.
<return value>	WideString	Returns the input value with all references to replaceable parameters replaced by the corresponding values.

This function parses the input value, replacing references to replaceable parameters with their corresponding values. Replaceable parameters are of the format **\$(<parameter name>,<format specifier>)** where the format specifier is optional. The following parameters are recognized:

Parameter Name	Description
Session.<property>	Where <property> is any property in the Session automation object.
Param.<name>	Where <name> is the name of a parameter created by a call to the <i>SetParam</i> method.
<object>.<property>	Where <property> is any property in the context object specified by <object>. Context objects provide a name when they are registered as a CSS service. This is the name used for the <object> specifier. For example, to access a patient's name, use the format PATIENT.NAME.
DEFDIR	The path to the current working directory.
WINDIR	The path to the windows directory.
SYSDIR	The path to the system directory.

This function is especially useful for setting properties of components when the values are not known at design time.

### ***UnregisterObject***

Parameter	Datatype	Description
ObjRef	IUnknown	IUnknown interface reference of the object to be unregistered.

The VIM uses this procedure call to remove event subscriptions for the object identified by the *ObjRef* interface reference.

## **ICSS SessionEvents**

This is the default outgoing interface for the Session automation object and is used to notify components of events of interest. The methods currently defined are:

### ***EventCallback***

Parameter	Datatype	Description
EventType	WideString	Identifies the type of event that has been signaled.
EventStub	WideString	Contains data describing details of the event that has been signaled. The format of this parameter is event specific.

The CSS invokes this callback when an event to which an object has subscribed has fired.

### ***RPCCallback***

Parameter	Datatype	Description
Handle	Integer	Unique handle of the remote procedure whose results are being returned.
Data	WideString	The return data of the remote procedure call in CommaText or PlainText format.

The CSS invokes this callback when an asynchronous RPC call has completed. The callback is made to the object that performed the asynchronous call. The *Handle* identifies which RPC is being reported (this is the value returned by the *CallRPCAsync* method of the Session automation object). *Data* represents any data returned by the RPC. If the *CallRPCAsync* method invoked the remote procedure with a *PlainText* parameter value of True, *Data* will be in plain text format (CR/LF-delimited), otherwise it is in comma text format.

### ***RPCCallbackError***

Parameter	Datatype	Description
Handle	Integer	Unique handle of the remote procedure generating the error.
ErrorCode	Integer	An error code value returned by the remote procedure.
ErrorText	WideString	A brief text message describing the error.

When an asynchronous RPC generates an exception, this callback method is called instead of the *RPCCallback* method.

## **ICSS Context**

This interface is defined by the CSS and must be implemented by every context object. The interface properties and methods allow the CSS to interact with the context object and make context change notifications on its behalf.

### ***Properties***

Property	Datatype	Access	Description
Callback	GUID	R	The GUID of the callback interface that will be used to notify subscribers of changes in this context object. Must be a descendant of <i>ICSS_ContextEvents</i> .
ContextName	WideString	R	The name by which this context will be advertised. This is the name that may be referenced in the <i>ReplaceParams</i> method of the Session object.
Pending	WordBool	R	If true, the context object has an uncommitted pending context.
Priority	Integer	R	Used to sequence processing of context. Context objects with higher priorities (lower values) are processed before those of lower priorities.

### ***CommitContext***

Parameter	Datatype	Description
Accept	WordBool	If true, the object should commit the pending context. If false, any pending context is cleared.

The CSS invokes this method to instruct a context object to commit or cancel its pending context.

### ***GetContext***

Parameter	Datatype	Description
Pending	WordBool	If true, the context object should return its pending context. Otherwise, the active context is returned.
<return value>	WideString	The active or pending context in CCOW format.

The CSS uses this method to request context information from the context object in preparation for initiating a CCOW context change. If the object does not produce a CCOW context, it should return a null string.

***Init***

The CSS invokes this method to instruct the context object to initialize itself to some default state. It is up to the context object to determine what default state to assume. For example, a patient context object might retrieve the last patient accessed by the current user.

***Reset***

The CSS invokes this method to instruct the context object to reset itself to a state that represents no context.

***SetContext***

Parameter	Datatype	Description
Context	WideString	Context the object is to assume, in CCOW format.
<return value>	WordBool	If true, the object successfully set its context. If false, the Context parameter contains no context information relevant to this object.

The CSS invokes this method to instruct the context object to initialize its pending context to conform to the context specified in Context. If the object is unable to comply, it should reset its pending context to a null state. If the Context parameter contains no context information relevant to the context object, the object should set its pending context to its default state and return false.

**User Context Object**

This interface defines properties and methods for accessing the global user object.

***Properties***

Property	Datatype	Access	Description
CanSignOrders	WordBool	R	Indicates whether or not the user can sign orders.
Esig	WideString	R	Returns the electronic signature code for the user. Reading this property causes the user to be prompted for their electronic signature code which, if valid, is then returned in encrypted form as the value of this property.
Handle	Integer	R	The host-specific handle identifying the user (a.k.a., DUZ).
IsProvider	WordBool	R	Indicates if the user belongs to a provider class.
Name	WideString	R	Full name of the user in the format Last, First, Middle.
NoOrdering	WordBool	R	If true, ordering is disabled on the host system.
OrderRole	Integer	R	Enumerated type indicating role of user in ordering process. Can be one of: 0 = none; 1 = clerk; 2 = nurse; 3 = physician; 4 = medical student; 5 = invalid data.
Service	Integer	R	The internal identifier for the service to which the user belongs
ServiceName	String	R	The external form of the service.
TimeOut	Integer	R	Idle timeout, in seconds.
UserClass	Integer	R	Enumerated type indicating ordering key possessed by user. Can be one of: 0 = none; 1 = OREMAS (ward clerk); 2 = ORELSE (nurse); 3 = ORES (physician).

***HasKey***

Parameter	Datatype	Description
KeyName	WideString	Name of the security key.
<return value>	WordBool	True if the user possesses the specified security key.

This function indicates whether or not the user has the specified security key.

***HasKeys***

Parameter	Datatype	Description
KeyNames	WideString	Multiple key names separated by circumflexes.
<return value>	WideString	Multiple Boolean values separated by circumflexes. Each value corresponds to a key name in the <i>KeyNames</i> parameter.

This function indicates whether or not the user has specified security keys. This function is useful when checking for the presence of multiple security keys. Since it requires only a single RPC call, it is more efficient than making one call for each key.

***ESigValidate***

Parameter	Datatype	Description
Signature	WideString	The user's unencrypted electronic signature code that is to be validated.
<return value>	WideString	If the Signature code is valid, the return value is the encrypted form of the electronic signature code. Otherwise, the return value is a null string.

This function validates the specific electronic signature code for the current user and returns the encrypted form of the code if the input is determined to be valid, or a null string otherwise.

## **Patient Context Object**

This interface defines properties and methods for accessing the global patient object.

### ***Properties***

Property	Datatype	Access	Description
AdmitDate	TDateTime	R	If an inpatient, this is the date and time of the current admission.
Age	Single	R	The patient's age, computed from the current date.
Attending	WideString	R	If an inpatient, the name of the patient's attending physician.
DOB	TDateTime	R	The patient's date of birth.
DOD	Date	R	Date of the patient's death.
Handle	Integer	RW	The host-specific handle identifying the patient (a.k.a., DFN). See description that follows for discussion of writable properties.
HRN	WideString	RW	Health record number for patient. See description that follows for discussion of writable properties.
ICN	WideString	RW	The patient's integration control number if one has been assigned. See description that follows for discussion about writable properties.
IsInpatient	WordBool	R	True if the patient is currently an inpatient.
IsRestricted	WordBool	R	
IsServiceConnected	WordBool	R	If true, the patient has a service-connected disability.
Name	WideString	R	The patient's full name, formatted as Last, First, Middle.
Location	Integer	R	If an inpatient, the internal identifier of the ward location.
LocationName	WideString	R	If an inpatient, the name of the ward location.
PercentServiceConnected	Integer	R	Returns the service-connected status of the patient as a percentage.
PrimaryProvider	WideString	R	The name of the patient's primary care provider.

PrimaryTeam	WideString	R	If an inpatient, the name of the patient's primary team.
RoomBed	WideString	R	If an inpatient, the room and bed number.
Sex	WideString	R	Indicates the patient's sex. One of: M = male; F = female; U = unknown.
Specialty	Integer	R	Treating specialty.
SSN	WideString	R	The patient's social security number, formatted as nnn-nn-nnnn.

Patient properties that are writable (*Handle* and *ICN*) may be conditionally modified by an application. This means that requesting a change to one of these properties is honored only if all context participants (both local and global) agree to the change. Therefore, an application must not assume that the change occurred but instead should either check the value after the assignment or wait for acknowledgement of the change (the *ICSS\_PatientEvents* event set).

## **Encounter Context Object**

This interface defines properties and methods for accessing the global encounter object.

***Properties***

Property	Datatype	Access	Description
DateTime	TDateTime	R	The date and time of the encounter.
Inpatient	WordBool	R	True if this is an inpatient encounter.
LocationName	WideString	R	The name of the encounter location.
LocationAbbr	WideString	R	The abbreviated name of the encounter location.
Location	Integer	R	The unique identifier of the encounter location.
Prepared	WordBool	R	True if a valid encounter has been recorded.
ProviderName	WideString	R	The name of the provider associated with the encounter.
Provider	Integer	R	The unique identifier of the provider associated with the encounter.
RoomBed	WideString	R	The room and bed # for an inpatient.
Standalone	WordBool	R	A true value indicates that this is a standalone encounter.
VisitCategory	Byte	R	Indicates the category of the visit. One of:
VisitID	WideString	R	The unique identifier for the encounter. (not currently implemented)
VisitStr	WideString	RW	A concatenation of the Location, DateTime, and VisitCategory properties.

This interface also implements the following method:

***Prepare***

Parameter	Datatype	Description
Filter	Integer	Applies the corresponding filter to the provider list.
<return value>	WordBool	True if a valid encounter was prepared.

Invokes the encounter dialog that allows the user to define an encounter context.

## **Site Context Object**

This interface defines properties and methods for accessing the global site object. This object describes the site that is currently connected.

### ***Properties***

Property	Datatype	Access	Description
Address1	WideString	R	First line of the facility's address.
Address2	WideString	R	Second line of the facility's address
City	WideString	R	City in which facility is located.
DomainName	WideString	R	The name of the domain.
Handle	Integer	R	The unique internal identifier for the facility.
FacilityID	WideString	R	The unique identifier for the facility.
LongName	WideString	R	The full name of the facility.
ShortName	WideString	R	The abbreviated name of the facility.
State	WideString	R	State in which facility is located.
ZipCode	WideString	R	Zipcode of the facility.

## **Context Change Events**

Each context object must declare a callback interface that is a descendant of the *ICSS\_ContextEvents* interface defined by the CSS. Though all such callback interfaces implement the identical methods, the GUIDs of each are unique to the context object that declares them. In this way, the CSS is able to notify subscribers of context change events on behalf of the respective context objects (because it declares and, therefore, understands the base interface), but is able to keep the subscriptions distinct. Components wishing to be notified of context changes must implement the callback interface declared by the context object of interest. Components should never implement the *ICSS\_ContextEvents* interface directly, but rather the descendant interface declared by the context object of interest (e.g., the *ICSS\_PatientEvents* interface if the patient context object is the target).

Note that because the method names are the same for all context change event interfaces (because they all have the same ancestor), components implementing more than one context

change interface must explicitly map the COM method names to the internal method names that implement them. The technique, called method aliasing, for accomplishing this varies by programming language.

Every context change callback interface declares the following methods:

### ***Pending***

Parameter	Datatype	Description
Silent	WordBool	If true, the component should not interact with the user to confirm the context change. This parameter will always be true if the context change request originated from the CCOW context manager.
<return value>	WideString	If the event subscriber wishes to contest a change in patient context, it should return a non-null string containing a brief description of the reason.

The CSS invokes this function whenever a request has been made to change the selected patient, but **before** the change has taken place. Subscribers wishing to participate in the decision to change the context for the corresponding context object may respond to this event by either prompting the user to save pending changes, warning the user that changes may be lost, and/or contesting the context change by returning a non-null value to the caller. Note, however, that if the Silent parameter is true, only the latter option should be exercised. A component should never request user interaction if the Silent parameter is true.

### ***Committed***

The CSS invokes this parameterless procedure after the pending context has been committed. Subscribers may respond by examining the corresponding context object and updating their state accordingly.

### ***Canceled***

The CSS invokes this parameterless procedure when a pending context change has been canceled. This can occur when a subscriber contests a pending change.

## Global Object Registry

The Object Registry provides information about components that are supported by VueCentric. Only components that are registered may be accessed. The object registry is stored in the *VUECENTRIC OBJECT REGISTRY* (#892.2) file on the host system. The fields in this file are:

Entity Name	Datatype	Description
ALLKEYS	Boolean	If true, the user must possess all keys listed in the KEYS multiple to access the object. Otherwise, possession of any one key is sufficient.
CATEGORY	Pointer	This is the category under which the object is to be classified. This controls where the object appears in the 'add object' dialog of the VIM design editor. Separate subcategories with the backslash character.
DESCRIPTION	Word Processing	A description of the object's purpose and any special procedures required for its implementation.
DISABLED	Boolean	If set to true, the object cannot be loaded by a configuration. Use this feature to take an object out of service.
HEIGHT	Integer	The default height, in pixels, when an object is created in design mode.
HIDDEN	Boolean	If true, the object does not appear in the Add Object dialog.
INITIALIZATION	String	These are the property initializers for an object. When an object is created, the properties listed here are initialized to the specified values. The format is <name>=<value>. Separate multiple initializers with a carriage return character.
KEYS	Pointer (multiple)	Security keys required to access this object. The ALLKEYS field determines whether all listed keys or any one key is required for access. If no keys are specified, access is unrestricted.
LOCATION	Pointer	The type of parent control the object is intended for. This affects where the object appears in the 'add object' dialog of the VIM design editor.
MULTIPLE	Boolean	If true, multiple instances of the object are allowed to exist concurrently in the same application instance.
NAME	String	This is a brief text description of the object. This is object name that is displayed by the 'add object' dialog.
PROGID	String	This is the programmatic identifier of the object and is the primary key for the file.
PROPEDIT	Boolean	If true, the object's internal property editor is invoked by the VIM rather than the default, generic property editor.
REGRESS	Boolean	If true, the object is retrieved from the repository if the repository version differs from the local version, even if the latter is newer.
REQUIRED	Word Processing	This is a list of URLs of additional files an object needs to run. For example, if an object requires a DLL to be installed, place a URL pointing to the

		DLL here. Separate multiple entries with a carriage return character.
SERVICE	Boolean	If true, the object represents a service that can be registered with the middle tier. Objects flagged as such are not displayed in the ‘add object’ dialog. If an object lists a service in its USES multiple, that service is automatically started when the object is loaded.
SIDEBYSIDE	Boolean	If true, objects are registered in such a manner as to support the co-existence of multiple versions of the object on the same workstation. This is useful in situation where multiple host systems are accessed, each with different client version requirements.
STREAMABLE	Word Processing	These are properties whose values are to be saved when a snapshot of the visual interface is taken. These property values are restored when the snapshot is loaded. Separate multiple property names with a carriage return character.
SOURCE	String	This is a URL which may be used to retrieve a copy of the object’s executable image. If no explicit path information is provided, the default path defined by the host system is used.
USES	Pointer (multiple)	This is a list of other objects that are required by this object. In contrast to entries in REQUIRED (which are only updated if the parent object is updated), the VIM performs version management on each entry in this multiple.
VERSION	String	This is the version of the object that is available from the URL named in SOURCE.
WIDTH	Integer	The default width, in pixels, when an object is created in design mode.

## Template Registry

A template is a snapshot of a visual interface. It contains all of the information required to reconstruct a visual interface including state information (like size, alignment, or color) and the parent-child relationships of the visual elements. Templates are used to create varied configurations of the VueCentric application and to create “compound objects” that can be dropped into the visual interface as if they were discrete objects.

Templates are stored in the *VUECENTRIC TEMPLATE REGISTRY* (#892.3) file. This file consists of two fields: a unique identifier and a word processing field.

When a user makes changes to his/her visual interface and saves them as his/her personal configuration, the VIM writes this information to the Template Registry. For user configuration templates, the template name begins with the '\$' character followed by the user's unique internal identifier. For application level templates, the template name begins with the '%' character. Both user and application templates differ from standard templates in that they also contain application level settings (e.g., default font, custom menus) whereas standard templates do not.

The format of state information varies by the type of associated object. Currently, eight stock object types (some of which are compound objects) are supported: object containers (TObjectContainer), panels (TPanelEx), scroll boxes (TScrollBoxEx), labels (TLabelEx), page controls (TPageControlEx / TTabSheetEx), toolbars (TToolbarEx), tree views (TTreeViewEx / TTreeViewPane) and splitter panes (TsplitterPaneEx / TPaneEx). Note that none of these are actually COM objects. Rather, they are specialized descendants of Delphi VCL controls. The actual COM objects are associated with and maintained by the object container (one per container). The object container is responsible for mediating the interaction between the contained COM object and the visual interface.

The entities stored under each registry node are as follows by object type:

### ***TObjectContainer***

Additional entities may be stored under an object container node that represents saved property values of the contained COM object. These entities always begin with an underscore to prevent name collisions with entities that pertain to the container itself. The object registry determines which properties of the contained object are saved.

Entity	Datatype	Description
CLASS	String	The class name of the control (TObjectContainer).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
PROGID	String	The programmatic identifier of the contained COM object.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.



***TPanelEx***

This is an implementation of a panel control upon which other controls may be placed.

Entity	Datatype	Description
CLASS	String	The class name of the control (TPanelEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

***TScrollBoxEx***

Similar to a panel control, this control automatically displays scrollbars if any control placed upon it is outside the current visual boundaries.

Entity	Datatype	Description
CLASS	String	The class name of the control (TScrollBoxEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

***TLabelEx***

This is a simple label that can be used to identify other components in the interface.

Entity	Datatype	Description
CLASS	String	The class name of the control (TPanelEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
CAPTION	String	The caption property of the control.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

***TToolbarEx***

This is a toolbar control that may have multiple controls (usually buttons) placed upon it. It automatically arranges the controls it contains.

Entity	Datatype	Description
CLASS	String	The class name of the control (TToolbarEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

***TPageControlEx***

This is a page control that can have multiple tabbed pages (*TTabSheetEx*) on it.

Entity	Datatype	Description
CLASS	String	The class name of the control (TPageControlEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
MULTILINE	Boolean	If true, the page control wraps tabs onto multiple lines if necessary. If false, scroll buttons appear if there are too many tabs to display within the current window dimensions.
PAGECOUNT	Integer	The number of tab sheets owned by the control.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the

		user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
TOPPAGE	Integer	The index of the tab sheet which is initially on top.
WIDTH	Integer	The width of the control in pixels.
TABPOSITION	Integer	The location of tabs on the page control. This is the equivalent of the TTabPosition enumerated datatype.

### ***TTabSheetEx***

These are the tab sheets that may appear on a page control.

Entity	Datatype	Description
CLASS	String	The class name of the control (TTabSheetEx).
CAPTION	String	The caption property of the control.
COLOR	TColor	The color of the associated tab.
PAGEINDEX	Integer	Order in which tab sheet appears on the parent control.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.

### ***TSplitterPaneEx***

This is a component with multiple panes separated by splitter bars that may be manually resized.

Entity	Datatype	Description
CLASS	String	The class name of the control (TSplitterPaneEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
BORDER	Boolean	If true, the control has a visible border
HEIGHT	Integer	The height of the control in pixels.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
ORIENTATION	Integer	The orientation of panes with the control. Possible

		values are 0 for horizontal and 1 for vertical.
PANECOUNT	Integer	The number of panes displayed by the control.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

### ***TPaneEx***

These are the individual panes that comprise a splitter pane control.

Entity	Datatype	Description
CLASS	String	The class name of the control (TPaneEx).
COLOR	TColor	The color of the associated pane.
HEIGHT	Integer	The height of the control in pixels.
PANEINDEX	Integer	The position of the pane within the parent control.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
WIDTH	Integer	The width of the control in pixels.

***TTreeViewEx***

This is a component with a tree view on one side and a pane view on the other. Each node of the tree has an associated pane that becomes visible in the pane view when the node is selected.

Entity	Datatype	Description
CLASS	String	The class name of the control (TTreeViewEx).
ALIGN	Integer	The alignment of the control relative to its parent. This is the equivalent of the TAlign enumerated datatype.
BORDER	Boolean	If true, the control has a visible border
DEFAULT	String	The path of the node whose pane is to appear when the control is initially loaded.
HEIGHT	Integer	The height of the control in pixels.
IMAGE	String	Specifies a file containing icon resources that are to be used by the control.
LEFT	Integer	The position of the leftmost portion of the control in the coordinate system of its parent.
ORIENTATION	Integer	The orientation of the control. If 0, the tree view is on the left. If 1, the tree view is on the right.
SIZE	Boolean	If true, the control display large icons.
SPLITTER	Integer	The position of the vertical splitter.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
TOP	Integer	The position of the topmost portion of the control in the coordinate system of its parent.
WIDTH	Integer	The width of the control in pixels.

***TTreePaneEx***

These are the individual panes that comprise a splitter pane control.

Entity	Datatype	Description
CLASS	String	The class name of the control (TTreePaneEx).
COLOR	TColor	The color of the associated pane.
IMAGE	Integer	The index of the icon to be displayed.
PATH	String	The path of the node. A path consists of the node caption preceded by the captions of each of its ancestor nodes, separated by backslashes.
TAG	Integer	The tag property of the control. If nonzero, the control and all its children cannot be modified by the user.
VISIBLE	Boolean	If true, the node is initially visible.

The following is a sample of a saved user configuration. Each node (whose parentage is represented as a sequence of three-digit hexadecimal numbers separated by backslashes) represents a specific visual element. The entities stored under each node represent the state of that visual element at the time the snapshot was taken. In the example below, the highlighted node is a container (TObjectContainer) for an ActiveX control. The actual ActiveX control is identified by the PROGID entity (CWPatientID.PatientID). Other entities denote state information about the object container. Entities starting with an underscore are property values of the contained ActiveX control.

```

\003\002\004\
CLASS=TPanelEx
TAG=0
LEFT=0
TOP=0
HEIGHT=52
WIDTH=1016
ALIGN=1
\003\002\004\000\
CLASS=TObjectContainer
TAG=0
LEFT=1
TOP=1
HEIGHT=50
WIDTH=247
ALIGN=3
PROGID=cwPatientID.PatientID
_COLOR=15780518

```

When reconstructing a saved configuration, the VIM performs a depth-first traversal of the subtree, instantiating the visual elements described by each node as it goes. The parent-child relationships represented in the subtree are reproduced as parent-child relationships in the visual interface. In the example above, the highlighted node has an ancestor that is a panel control. The panel control itself has two ancestors (as can be ascertained from the node path), though their identities are not shown in this example.

## Object Repository

The Object Repository provides a centralized location for storing the most up-to-date versions of VueCentric components. The Object Repository may be implemented on a web server, an FTP server, a shared network directory, or any combination of these. The Object Repository works in concert with the Object Registry to permit the automatic updating of components. The Object Registry provides information about the components stored in the Object Repository including version information and a URL to be used to locate an updated version of a component.

Typically a site will implement its Object Repository in one of the three locations mentioned. However, it is entirely possible that a site may implement components that are developed and maintained by another site. In such a scenario, it would be logical to retrieve updates to such a component directly from the originating site, typically using the FTP or HTTP protocol. However, the implementing site must still update its Object Registry to indicate when a new version is available.

When a component is requested by the VIM, which can occur in design mode when a component is dropped into the visual interface or during the loading of a saved configuration, the VIM checks the locally installed version of the component with the version available from the Object Repository. If the Object Repository has a newer version, or if the component has not yet been installed on the local machine, the VIM downloads a copy of the component from the Object Repository (using the URL specified in that component's Object Registry entry). It then automatically registers the updated component before instantiating it within the interface. Other than a slightly perceptible delay while the download occurs, this process is essentially transparent to the user and occurs without direct intervention.

# Glossary

**ActiveX object** – A COM object that implements specific standard interfaces.

**Component Object Model (COM)** – A Microsoft specification that defines an architecture that permits the interoperability of objects independent of their implementation.

**COM object** – An object that complies with Microsoft's Component Object Model.

**Connection point** – An outgoing interface on a server object to which a client may connect to receive event notifications. If the server object supports it, multiple clients may concurrently connect to the same connection point.

**Dispatch interface** – A special kind of interface whose methods and properties may be discerned and accessed at run-time. Also known as an automation interface, this kind of interface is used to control objects from languages that support late binding (e.g., script languages like VB Script).

**Dual interface** – An interface that implements a dispatch interface and a standard v-table interface.

**GUID** – (Globally unique identifier) This is an identifier assigned to COM objects, interfaces, and type libraries that is guaranteed (at least, statistically) to be unique across all systems.

**Incoming interface** – A collection of methods and properties implemented by an object. Clients of the implementing object invoke its incoming interface methods.

**OLE object** – Synonymous with COM object.

**Outgoing interface** – Objects do not implement outgoing interfaces. Rather they indicate their ability to act a client for the interface to objects that do implement it. Server objects often use outgoing interfaces to talk back to their clients (and in that capacity, the role of client and server is reversed).

**Programmatic identifier** – This is a brief, text descriptor of a COM object. While uniqueness is not guaranteed for programmatic identifiers, it is frequently advantageous to refer to an object by its programmatic identifier rather than its GUID.

# MIDL Specification

The following are the Microsoft IDL specifications for the principal VueCentric interfaces.

## Visual Interface Manager

```
[ uuid(C1FD48D2-2C3B-42B4-AB77-AD90F8C7BA8B),
  version(3.0),
  helpstring("Visual Interface Manager Library")
]
library CIA_VIM
{
  importlib("stdole2.tlb");
  importlib("CSS.dll");
  importlib("CSSUser.dll");
  importlib("STDVCL40.DLL");
  [ uuid(B96FFF00-E72A-4C9C-BBEA-F0C93864BE3D),
    version(1.0),
    helpstring("Dispatch interface for controlling VIM"),
    dual,
    oleautomation
  ]
  interface IVIM: IDispatch
  {
    [propget, id(0xFFFFFDF5), helpstring("Sets the text that appears in the application's title bar.")]
    HRESULT _stdcall Caption([out, retval] BSTR * Value );
    [propput, id(0xFFFFFDF5), helpstring("Sets the text that appears in the application's title bar.")]
    HRESULT _stdcall Caption([in] BSTR Value );
    [propget, id(0xFFFFFE00), helpstring("Sets the default font for the application.")]
    HRESULT _stdcall Font([out, retval] IFontDisp ** Value );
    [propput, id(0xFFFFFE00), helpstring("Sets the default font for the application.")]
    HRESULT _stdcall Font([in] IFontDisp * Value );
    [propget, id(0xFFFFFDF5), helpstring("Sets the background image that appears when the application is logged out.")]
    HRESULT _stdcall Picture([out, retval] BSTR * Value );
    [propput, id(0xFFFFFDF5), helpstring("Sets the background image that appears when the application is logged out.")]
    HRESULT _stdcall Picture([in] BSTR Value );
    [propget, id(0x00000001), helpstring("Sets the icon associated with the application.")]
    HRESULT _stdcall Icon([out, retval] BSTR * Value );
    [propput, id(0x00000001), helpstring("Sets the icon associated with the application.")]
    HRESULT _stdcall Icon([in] BSTR Value );
    [id(0x00000002), helpstring("Brings the specified object to the foreground")]
    HRESULT _stdcall BringToFront([in] IUnknown * ObjRef, [out, retval] VARIANT_BOOL * Value );
    [id(0x00000003), helpstring("Pops up the specified object in a modal window")]
    HRESULT _stdcall Popup([in] IUnknown * ObjRef, [in] BSTR Title, [out, retval] VARIANT_BOOL * Value );
  };
  [propget, id(0x00000005), helpstring("Provides access to the VIM status bar")]
  HRESULT _stdcall StatusBar([in] long Index, [out, retval] BSTR * Value );
  [propput, id(0x00000005), helpstring("Provides access to the VIM status bar")]
  HRESULT _stdcall StatusBar([in] long Index, [in] BSTR Value );
};
[ uuid(A45DCEDF-22F6-4F2B-BA12-DF5D5765ED68),
  version(1.0),
  helpstring("VIM Object")
]
coclass VIM
{
  [default] interface IVIM;
  interface ICSS_SessionEvents;
  interface ICSS_UserEvents;
};
};
```

## Component Support Services

```
[ uuid(8B8F116E-35A6-4654-8E22-DAAFDDB1839A7),
  version(4.0),
  helpstring("Component Support Services")
]
library CIA_CSS
{ importlib("stdole2.tlb");
  importlib("STDVCL40.DLL");

  [ uuid(8448917E-D5A0-43A3-9B58-F964D7ACCC49),
    version(1.0),
    helpstring("Dispatch interface for Session Object"),
    dual,
    oleautomation
  ]
  interface ICSS_Session: IDispatch
  { [id(0x00000001)]
    HRESULT _stdcall Connect([in] BSTR HostName, [out, retval] VARIANT_BOOL * Value );
    [id(0x00000002)]
    HRESULT _stdcall Disconnect([in] VARIANT_BOOL Survey, [out, retval] VARIANT_BOOL * Value );
    [propget]
    HRESULT _stdcall HostName([out, retval] BSTR * Value );
    [propget, id(0x00000004)]
    HRESULT _stdcall HostPort([out, retval] long * Value );
    [propget, id(0x00000005)]
    HRESULT _stdcall CCOWState([out, retval] EnumCCOWState * Value );
    [id(0x00000006)]
    HRESULT _stdcall CCOWJoin([out, retval] VARIANT_BOOL * Value );
    [id(0x00000007)]
    HRESULT _stdcall CCOWLeave( void );
    [id(0x00000008)]
    HRESULT _stdcall CallRPCList([in] BSTR RPCName, [in] VARIANT Params, [out, retval] BSTR * Value );
    [id(0x00000009)]
    HRESULT _stdcall CallRPCText([in] BSTR RPCName, [in] VARIANT Params, [out, retval] BSTR * Value );
    [id(0x0000000A)]
    HRESULT _stdcall CallRPCStr([in] BSTR RPCName, [in] VARIANT Params, [out, retval] BSTR * Value );
    [id(0x0000000B)]
    HRESULT _stdcall CallRPCInt([in] BSTR RPCName, [in] VARIANT Params, [out, retval] long * Value );
    [id(0x0000000C)]
    HRESULT _stdcall CallRPCBool([in] BSTR RPCName, [in] VARIANT Params, [out, retval] VARIANT_BOOL *
Value );
    [id(0x0000000D)]
    HRESULT _stdcall CallRPCDate([in] BSTR RPCName, [in] VARIANT Params, [out, retval] DATE * Value );
    [id(0x0000000E)]
    HRESULT _stdcall CallRPCAsync([in] BSTR RPCName, [in] VARIANT Params, [in] ICSS_SessionEvents *
CallBack, [in] VARIANT_BOOL PlainText, [out, retval] long * Value );
    [id(0x0000000F)]
    HRESULT _stdcall CallRPCAbort([in] long Handle );
    [id(0x00000010)]
    HRESULT _stdcall RegisterObject([in] IUnknown * ObjRef );
    [id(0x00000011)]
    HRESULT _stdcall UnregisterObject([in] IUnknown * ObjRef );
    [id(0x00000012)]
    HRESULT _stdcall ReplaceParams([in] BSTR Source, [out, retval] BSTR * Value );
    [id(0x00000013)]
    HRESULT _stdcall FindObjectByCLSID([in] GUID CLSID, [in] IUnknown * Last, [out, retval] IUnknown **
Value );
    [id(0x00000014)]
    HRESULT _stdcall FindObjectByProgID([in] BSTR ProgID, [in] IUnknown * Last, [out, retval] IUnknown
** Value );
    [id(0x00000015)]
    HRESULT _stdcall FindObjectByIID([in] GUID IID, [in] IUnknown * Last, [out, retval] IUnknown **
Value );
    [id(0x00000016)]
    HRESULT _stdcall FindServiceByCLSID([in] GUID CLSID, [out, retval] IUnknown ** Value );
    [id(0x00000017)]
    HRESULT _stdcall FindServiceByProgID([in] BSTR ProgID, [out, retval] IUnknown ** Value );
    [id(0x00000019)]
    HRESULT _stdcall EventSubscribe([in] BSTR EventType, [in] ICSS_SessionEvents * CallBack );
```

```

[id(0x0000001A)]
HRESULT _stdcall EventUnsubscribe([in] BSTR EventType, [in] ICSS_SessionEvents * Callback );
[id(0x0000001B)]
HRESULT _stdcall EventFireLocal([in] BSTR EventType, [in] BSTR EventStub );
[id(0x0000001C)]
HRESULT _stdcall EventFireRemote([in] BSTR EventType, [in] BSTR EventStub, [in] BSTR Recipients );
[propget, id(0x0000001D)]
HRESULT _stdcall DebugMode([out, retval] VARIANT_BOOL * Value );
[propput, id(0x0000001D)]
HRESULT _stdcall DebugMode([in] VARIANT_BOOL Value );
[propget, id(0x0000001E)]
HRESULT _stdcall HostDateTime([out, retval] DATE * Value );
[id(0x0000001F)]
HRESULT _stdcall ContextChangeBegin( void );
[id(0x00000020)]
HRESULT _stdcall ContextChangeEnd( void );
[propget, id(0x00000023)]
HRESULT _stdcall DomainName([out, retval] BSTR * Value );
};

[ uuid(8010CCA9-D6B6-4C0D-8618-3D96456486E2),
  version(1.0),
  dual,
  oleautomation
]
interface ICSS_SessionEvents: IDispatch
{ [id(0x00000001)]
  HRESULT _stdcall RPCCallback([in] long Handle, [in] BSTR Data );
  [id(0x00000002)]
  HRESULT _stdcall RPCCallbackError([in] long Handle, [in] long ErrorCode, [in] BSTR ErrorText );
  [id(0x00000003)]
  HRESULT _stdcall EventCallback([in] BSTR EventType, [in] BSTR EventStub );
};

[ uuid(C1FDFB04-000C-48A0-97E6-6E7D5861003E),
  version(1.0),
  helpstring("Session Object"),
  noncreatable
]
coclass CSS_Session
{ [default] interface ICSS_Session;
  [default, source] interface ICSS_SessionEvents;
};

[ uuid(C4B85076-DC45-4470-9E7C-5471ED7074DC),
  version(1.0),
  dual,
  oleautomation
]
interface ICSS_Server: IDispatch
{ [propget, id(0x00000001)]
  HRESULT _stdcall Session([out, retval] ICSS_Session ** Value );
};

[ uuid(8C061A95-8FCE-41A7-A806-66B02E5CE6EF),
  version(1.0)
]
coclass CSS_Server
{ [default] interface ICSS_Server;
};

[ uuid(3414C32D-952D-4529-96B4-A96FD60BB8B9),
  version(1.0),
  dual,
  oleautomation
]
interface ICSS_Context: IDispatch
{ [propget, id(0x00000001)]
  HRESULT _stdcall ContextName([out, retval] BSTR * Value );
  [id(0x00000002)]
  HRESULT _stdcall GetContext([in] VARIANT_BOOL Pending, [out, retval] BSTR * Value );
};

```

```

[id(0x00000003)]
HRESULT _stdcall SetContext([in] BSTR Context, [out, retval] VARIANT_BOOL * Value );
[id(0x00000004)]
HRESULT _stdcall CommitContext([in] VARIANT_BOOL Accept );
[propget, id(0x00000005)]
HRESULT _stdcall Pending([out, retval] VARIANT_BOOL * Value );
[propget, id(0x00000006)]
HRESULT _stdcall Priority([out, retval] long * Value );
[propget, id(0x00000007)]
HRESULT _stdcall CallBack([out, retval] GUID * Value );
[id(0x00000008)]
HRESULT _stdcall Init( void );
[id(0x00000009)]
HRESULT _stdcall Reset( void );
};

[ uuid(5A180A55-1175-4018-9F70-819039496C46),
  version(1.0),
  dual,
  oleautomation
]
interface ICSS_ContextEvents: IDispatch
{ [id(0x00000001)]
  HRESULT _stdcall Pending([in] VARIANT_BOOL Silent, [out, retval] BSTR * Value );
  [id(0x00000002)]
  HRESULT _stdcall Committed( void );
  [id(0x00000003)]
  HRESULT _stdcall Canceled( void );
};

[ uuid(911F8FF9-A941-42EF-8BCB-E87B8C87F71F),
  version(1.0)
]
typedef enum tagEnumCCOWState
{ ccowBroken = 0,
  ccowChanging = 1,
  ccowJoined = 2,
  ccowNone = 3,
  ccowDisabled = 4
} EnumCCOWState;
};

```